DTIC FILE COPY

(4)

Uncla
SECURITY CL

1. REPORT # **AD-A204 256** CCESSION NO.	READ INSTRUCTIONS BEFORE COMPLETING FORM 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Taxonomy of Synchronous Parallel Machines	5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Lawrence Snyder	8. CONTRACT OR GRANT NUMBER(s) N00014-86-K-0264
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Washington Department of Computer Science Seattle, Washington 98195	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Information Systems Program Arlington, VA 22217	12. REPORT DATE August 1988 13. NUMBER OF PAGES 5
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

DTIC ELECTE JAN 23 1989 S D CH

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

reference stream, Flynn's taxonomy, synchroneity, computer architecture, synchronous parallel computers

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A new classificational scheme is presented which is consistent with Flynn's taxonomy but is more expressive. The crucial idea is to recognize that a reference stream is composed of both values and addresses; their treatment exposes critical features of an architecture. This insight, together with the accompanying formal mechanism built on top of it, enables a large variety of recently developed (since Flynn's work) machines to be distinguished, including VLIW, multigauge, systolic arrays, and the Connection Machines. Though the resulting taxonomic structure is illuminating, the most important result

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (continued)

of the classification is the discovery that synchronous execution is NOT a defining property of computer architectures, but is a derived property, a consequence of other architectural features. The evidence for this result and the consequences for machine classification are presented.

A Taxonomy of Synchronous Parallel Machines

Lawrence Snyder
Department of Computer Science
University of Washington
Seattle, Washington 98195[1]

Abstract: A new classificational scheme is presented which is consistent with Flynn's taxonomy but is more expressive. The crucial idea is to recognize that a reference stream is composed of both values and addresses; their treatment exposes critical features of an architecture. This insight, together with the accompanying formal mechanism built on top of it, enables a large variety of recently developed (since Flynn's work) machines to be distinguished, including VLIW, multigauge, systolic arrays, and the Connection Machines. Though the resulting taxonomic structure is illuminating, the most important result of the classification is the discovery that synchronous execution is NOT a defining property of computer architectures, but is a derived property, a consequence of other architectural features. The evidence for this result and the consequences for machine classification are presented.

1 Introduction

In 1966 Flynn [1] introduced his classification of computers. This taxomony proved to be very useful, giving us terminology like SIMD and MIMD that endures to this day. The taxonomy, however, has long been described as too coarse, unable to distinguish between computers that seem to computer architects to be quite different. Though other classifications have been offered [2-5], the fact that Flynn's classification has lasted for so long without being replaced and enhanced is a testament to the difficulty of discovering something better.

In this paper a new taxonomy is presented for synchronous parallel computers. It has no pretentions of being complete nor of capturing all features of synchronous parallel computers. The taxonomy does clarify important distinctions among recently developed parallel computers, such as the VLIW machines, multigauge machines and certain SIMD machines such as the Connection Machines.

The key idea of the taxonomy is to quantify the components of the fetch/execute cycle that process I-streams and D-streams. To make fine distinctions among machines, one must separate these reference streams into their address and value components, because addressing and value processing are crucial features by which machines differ.

Using this kind of analysis, a taxonomy is constructed. Many of the machines that are placed into different classes here would have been classified by Flynn's scheme as SIMD, so this approach permits finer distinctions to be made. Only a small number of classes have been described, and only one or two machines per class have been identified. Thus, there remain substantial opportunities for further research.

[1]This research funded in part by the Office of Naval Research Contract N00014-86-K-0264, National Science Foundation Grant CCR-8416678 and Air Force Office of Scientific Research Contract 88-0023.

Perhaps the most important result derived from the taxonomy concerns the property of "synchroneity". The author and apparently many other researchers have treated synchroneity as a primary classificational property: we have spoken of "the synchronous vs. the asynchronous" machines as if this should be an important way to distinguish between machines. It is not. The criterion used for classifying machines in this taxonomy tells when a machine must have all of its instructions start at the same time, and when it is not necessary. This determination is based on how the machine addresses and processes instructions and data. Machines which must begin all instructions at the same time will automatically be synchronous; for those machines that need not begin their instructions at the same time it is an "engineering decision" whether to make them synchonous or asynchronous. Thus the quality of being synchronous is a derived property: A machine must have it because of other features, or it is a noncritical implementation feature.

2 Preliminaries

A *reference stream*, S, of a computer is a finite set of infinite sequences of pairs,

$$S = \{ (a_1 < t_1 >)(a_2 < t_2 >) \ldots, (b_1 < u_1 >)(b_2 < u_2 >) \ldots, \ldots, (c_1 < v_1 >)(c_2 < v_2 >) \ldots \}$$

the first component of each pair being a nonnegative integer, called an *address*, and the second component being an n-tuple of nonnegative integers, called *values*, such that n is the same for all tuples of all sequences. An element of a reference stream is called a *reference sequence*. An *I-stream* is a reference stream whose values are interpreted as *instructions*; a *D-stream* is a reference stream whose values are interpreted as *data*.

The interpretation of these definitions is simple. The elements of reference sequences are address, value pairs, the values simply being the contents fetched from (or stored to) memory at the address. A sequence of elements can be thought of as the history of the addresses and values moving between a processor and its memory space. An I-stream is made up of a finite set of these sequences, the number depending on how many instruction sequences the machine can process at one time; and a D-stream is made of a finite set of data sequences, the number depending on how many distinct operations the machine can perform at one time.

Although the I-streams and D-streams have been defined in an intuitive manner, their form is not convenient

for analysis. Accordingly, the following reassociation must be performed. Let

$$S = \{ (a_1 < t_1 >)(a_1 < t_2 >) \ldots,$$
$$(b_1 < u_1 >)(b_2 < u_2 >) \ldots \ldots \ldots,$$
$$(c_1 < v_1 >)(c_2 < v_2 >) \ldots \}$$

be a reference stream. Define two sequences: The *address sequence of S*, denoted S_a, is a sequence whose i^{th} element is a tuple formed from the addresses from the i^{th} elements of each sequence of S,

$$S_a = < a_1 b_1 c_1 >, < a_2 b_2 c_2 >, \ldots$$

and the *value sequence of S*, denoted S_v, is the sequence whose i^{th} element is a tuple formed by concatenating the value tuples from the i^{th} elements of each reference sequence of S,

$$S_v = < t_1 u_1 v_1 >, < t_2 u_2 v_2 >, \ldots .$$

Notice that although a reference stream is a set of sequences, address and value sequences are just sequences of tuples.

It is possible to interpret these definitions as grouping the corresponding addresses and corresponding value tuples of a reference stream S into S_a and S_v, respectively.

Let S_x be a sequence of n-tuples; the *width* of the sequence, $w(S_x) = n$.

Proposition 1: Let S be a reference stream with n-tuple values, then

$$w(S_a) = \mid S \mid \text{ and } w(S_v) = n \mid S \mid$$

where $\mid X \mid$ denotes the cardinality of the set X.

A *computation* is a pair (I,D), where I is an I-Stream and D is a D-stream. Computers are classified by the computations they execute. A computer *executes* the computation (I,D) provided it presents $w(I_a)$ instruction addresses to memory to be fetched simultaneously, it decodes and interprets $w(I_v)$ instructions simultaneously, it presents $w(D_a)$ operand addresses to memory simultaneously, and it performs $w(D_v)$ operations on distinct data values simultaneously. The computer is described by the notation

$$I_{w(I_a)w(I_v)}\, D_{w(D_a)w(D_v)}.$$

Notice that we speak of *the* computation executed by a computer. This is a definitional simplification, and is sufficient since any desired sequence of instructions or data is a subsequence of the infinite streams of the computation. Observe the relationship between this point and the Enumeration Theorem of recursive function theory.

Let d_1, d_2, d_3 and d_4 be predicates called *class designators*; then a machine is said to be member of the *class* denoted by

$$I_{d_1 d_2} D_{d_3 d_4}$$

if and only if $d_1(w(I_a)), d_2(w(I_v)), d_3(w(D_a))$, and $d_4(w(D_v))$. (Commas may occasionally be inserted between the subscripts for clarity.)

Example 2: By appropriating for our class designators Flynn's "s" and "m" to denote the predicates *"is-one"* and *"is-many"*, it is possible to classify some familiar machines using the mechanisms developed so far.

Let a von Neumann machine, which Flynn classified as SISD, execute the computation (I,D). From his classification we have

$$\mid I \mid = \mid D \mid = 1.$$

By Proposition 1, then, we have

$$w(I_a) = w(D_a) = 1.$$

Moreover, since instructions are decoded serially, $w(I_v) = 1$ and since they are executed serially, $w(D_v) = 1$. Therefore the von Neumann machine is described as

$$I_{1,1} D_{1,1}.$$

It is classified with the present notation as

$$I_{ss} D_{ss}$$

since the predicate "s"is true for all four widths.

Now consider two machines that Flynn's taxonomy lumped in the SIMD category, the MPP and the Illiac IV. (Ignore for the moment the fact that these have bit serial and word parallel PEs, respectively.) The single instruction stream means $\mid I \mid = 1$ for both machines. By the same reasoning just used for the von Neumann machine, the instruction streams for both machines are described as I_{ss}.

For data, consider the MPP first. Recall that the MPP controller broadcasts the same data memory address to all PEs [6], and so the machine has a single D-stream in *our* terminology; $\mid D \mid = 1$. However, a value is fetched from each PE memory, so the values of this stream are 16384-tuples. Thus,

$$w(D_a) = 1 \text{ and } w(D_v) = 16384,$$

which certainly satisfies the "multiple" class designator. So, the MPP is described as

$$I_{1,1} D_{1,16384}$$

and is classified as

$$I_{ss} D_{sm}.$$

The MPP has a "multiple data stream" but the multiplicity applies only to the data values, not to the data value addressing.

For the Illiac IV on the other hand, the controller broadcasts a base address to all PEs, each of which may produce its own address by adding in the contents of a local index register [7]. This means that $\mid D \mid = 64$; there are 64 operand address streams simultaneously produced by the machine and each of them references a single value, i.e. each data address is associated with a 1-tuple. Accordingly,

$$w(D_a) = w(D_v) = 64$$

and the Illiac IV is described as

$$I_{1,1}\ D_{64,64}$$

which places it in the

$$I_{ss}D_{mm}$$

class. It has "multiple data streams" too, but its multiplicities are for addressing and data reference. Clearly, the present taxonomy retains the distinctions achieved by Flynn, but it is also capable of making finer distinctions.

3 Discussion

It is possible to give an intuitive interpretation to much of the foregoing formalism. The key idea is to recognize that the formalism quantifies funtional components of a fetch/execute cycle. Thus, the machine described as

$$I_{av}D_{a'v'}$$

presents instruction addresses to memory for a threads of control (presumably from a PCs but data flow computers qualify as well); it receives v different instructions back from memory at once and interprets them; it presents a' different operand addresses to memory for data values, and it receives v' data values back and operates upon them concurrently. So, when the MPP is described as

$$I_{1,1}D_{1,16384}$$

it is immediately obvious that its PEs all use the same address for accessing their operand values, even though they are capable of independently performing operations on the resulting data.

The interpretation of the classification is intended to carry the implication that if the n-tuple of values $< t_i >$ is received from memory upon presentation of address a_i then the machine is capable of processing all n elements at once. This applies to both instructions and data. So even if a computer makes a memory reference to address a_i and fetches k words, perhaps to cache them, if it only processes one of them, then $n = 1$ in this model.

Finally, notice that our classificational scheme is a completely formal system with a precise meaning. Its utility in classifying computers depends entirely on our interpreting this formalism as meaningful. Though it is possible for two scientists to differ in their interpretation, and thus to differ on a classification, the underlying scheme is unambiguous.

4 Properties of Address and Value Sequences

To simplify discussing computer families, it is convenient to adopt a simple abbreviation. The expression

$$I_{d_1d_2}D_{d_3d_4}$$

will be abbreviated by the string

$$d_1d_2d_3d_4$$

Thus, the von Neumann machine class is abbreviated *ssss*, while the MPP is in *sssm*. String expressions will be used as shorthand to abbreviate several classes.

There are several important properties of this taxonomic system which influence the kind of machine classes definable.

Proposition 3: Any machine $I_{av}D_{a'v'}$ satisfies the inequalities:

$$a \leq v \text{ and } a' \leq v'$$

These inequalities follow from the fact that in a reference stream every address is paired with at least one value, so the width of the address stream is a lower limit to the width of the corresponding value stream. The interpretation of these inequalities seems intuitively correct: The number of addresses presented to memory should never exceed the number of values returned. As a corollary, any nonempty machine class will satisfy these relationships, where the definition of the relation is suitably extended.

Convention 4: Any machine $I_{av}D_{a'v'}$ will satisfy the inequality:

$$v \leq v'$$

Unlike the preceding propositions which are artifacts of the taxonomy's abstraction, this convention is adopted primarily for semantic consistancy. Its interpretation is that the number of instructions being interpreted should not exceed the data available.

Since it is a convention, it is open to debate. On the positive side the convention helps avoid "problem" machines like Flynn's MISD; this machine doesn't make much sense and has often been criticized. Here, the convention is worthwhile, considering that the finer control of this taxonomy permits greater opportunities to create such dubious classes. On the negative side, adopting the convention might prevent accurately describing certain machines, though none has come to the author's attention. Since a taxonomy is descriptive (as opposed to being prescriptive) and given that architects are not likely to have their creativity constrained by this convention, we adopt it.

5 More Machine Classes

The efficacy of a classificational system usually depends to some degree on interpretation. (It always does in biological taxonomy.) Usually there is a large range (sometimes a continuum) of values that a property can assume, and we wish to assign certain segments of this range to different classes. But there may not be any effective way to identify the boundaries of these ranges, and so membership is often a matter of judgement. This characteristic will persist for this taxonomy, but confusion can be minimized by being somewhat more precise about the terminology that we've already used.

Define the class designators as follows:

- s is the predicate "equals 1",
- c is the predicate "from 1 to some (small) constant", and
- m is the predicate "from 1 to an arbitrarily large finite number".

Though the c and m designators have no upper limit in principle, they are intended to convey two different meanings. When the c designation is used the range has a hard upper limit usually due to internal constraints in the architecture and cannot be easily increased by a substantial amount. An example might be the number of instructions that can be packed in the instruction word of a VLIW machine[8]; for any given word size it is fixed, and even though the word size can be increased this is probably not the intended nor the rational way to generalized the given machine. The m designation, however, is used when the quantity can be easily generalized or scaled. An example is when additional PEs can be added as with the MPP.

These distinctions are not always clear, of course, and judgement must be applied. An example is the question of how to classify a machine with processors connected to a bus [9]. In principle, there is no limit to how many processors can be attached to a bus, but with the addition of each processor the congestion increases, and this is an internal constraint reducing the performance. Is this a "c" or "m" case? Arguments can be made on both sides; we leave the question open for the moment.

It is now possible, using the class designators, Proposition 3 and the convention to define a number of machine classes. Notice that there is no attempt to be complete in either defining classes or categorizing machines:

$I_{ss}D_{ss}$ von Neumann machines.

$I_{ss}D_{sc}$ "packed" von Neumann[10]; the machine can fetch several distinct data values from fixed postions from one address and simultaneously apply the same operation to them. Many machines have some instructions of this form, e.g. performing 2 half word adds on the word at a given address; all (ALU) instructions for a machine in this class would have this capability.

$I_{ss}D_{sm}$ SIMD Parallel Machines with no addressability, such as the MPP, the Connection Machine 1 [11] and systolic arrays [12].

$I_{ss}D_{cc}$ SIMD Multigauge machines [13]; these are von Neumann machines which can (optionally) split their datapath to process multiple, independent operand streams at once.

$I_{ss}D_{mm}$ Addressable SIMD Parallel Machines, such as Illiac IV and the CM2 [14].

$I_{sc}D_{cc}$ VLIW Machines [8]; the machine fetches and executes several instructions stored in one instruction address.

$I_{cc}D_{cc}$ MIMD Multigauge machines [13]; these are von Neumann machines which can (optionally) split their fetch/execute cycle to process multiple, independent instructions concurrently.

$I_{mm}D_{mm}$ MIMD Parallel Machines, including machines such as the Ultracomputer[15] and the Cosmic Cube[16].

Clearly, the list is not complete in terms of either the classes listed or the machines recognized as members of any given class. Much work remains.

6 Discussion of the Taxonomy and The Origins of Synchronous Computation

One is struck by at least two aspects of the foregoing classification: A large and diverse set of machines are lumped into the last classification, *mmmm*, and nowhere in the taxonomy has the synchronous requirement been mentioned, except in the paper's title. These two observations are related.

In effect the taxonomy uses as its "criterion for classification" the number of repeated instances of the principle functional activities of the fetch/execute cycle. So, machines are distinguished by how many instructions they can decode at once or how many operations on separate data they can perform simultaneously. But these are not the characteristics we think of as distinguishing the different MIMD parallel computers. Rather, we think of them as being different depending on whether or not they have global shared memory or what their interconnection topology is. These are features unrelated to the fetch/execute cycle. So, lumping MIMD parallel computers in the *mmmm* class says only that by the criterion applied, they are all equivalent.

This is unsurprising and is not evidence of weakness in the classification. Indeed, it might point to why efforts to find criteria suitable for classifying all parallel machines have so far been unsuccessful: Qualities that are important for some computers are for other computers, unimportant, irrelevant or even misleading as a guide to classification. To the extent that the taxonomy provides insight by its classifications, the "criterion" amounts to being a useful way of looking at some computers. (For cases where topology matters for sychronous machines, e.g. between the MPP and the CM1, see the next section.)

Interestingly, the "criterion" apparently mandates the synchronous property. By using "the number of repeated instances of the principle functional activities of the fetch/execute cycle" as the basis for classification, we are thinking of machines as either a single f/e cycle that has certain components replicated, or multiple copies of a f/e cycle. In the former case (all legal machines of the form $sy.\ y \in \{s,c,m\}^3$) synchronous execution is mandated because there is only one cycle running. With the current structure of the taxonomy this leaves only the cy classes and *mmmm*. Though there is no requirement in the model that these be synchronous, the class designators provide some basis for deciding: The c designation carries with it the implication